A Decision Procedure for (Co)datatypes in SMT Solvers

Andrew Reynolds, Jasmin Christian Blanchette

A Decision Procedure for (Co)datatypes in SMT Solvers

Andrew Reynolds[1] and Jasmin Christian Blanchette[2,3]

[1] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
[2] Inria Nancy & LORIA, Villers-lès-Nancy, France
[3] Max-Planck-Institut für Informatik, Saarbrücken, Germany

In memoriam Morgan Deters 1979–2015

Abstract. We present a decision procedure that combines reasoning about datatypes and codatatypes. The dual of the acyclicity rule for datatypes is a uniqueness rule that identifies observationally equal codatatype values, including cyclic values. The procedure decides universal problems and is composable via the Nelson–Oppen method. It has been implemented in CVC4, a state-of-the-art SMT solver. An evaluation based on problems generated from theories developed with Isabelle demonstrates the potential of the procedure.

1 Introduction

Freely generated algebraic datatypes are ubiquitous in functional programs and formal specifications. They are especially useful to represent finite data structures in computer science applications but also arise in formalized mathematics. They can be implemented efficiently and enjoy properties that can be exploited in automated reasoners.

To represent infinite objects, a natural choice is to turn to coalgebraic datatypes, or *codatatypes*, the non-well-founded dual of algebraic *datatypes.* Despite their reputation for being esoteric, codatatypes have a role to play in computer science. The verified C compiler CompCert [13], the verified Java compiler JinjaThreads [14], and the formalized Java memory model [15] all depend on codatatypes to capture infinite processes.

Codatatypes are freely generated by their constructors, but in contrast with datatypes, infinite constructor terms are also legitimate values for codatatypes (Section 2). Intuitively, the values of a codatatype consist of all well-typed finite and infinite ground constructor terms, and only those. As a simple example, the coalgebraic specification

$$\textbf{codatatype}\ \mathit{enat} = \mathsf{Z} \mid \mathsf{S}(\mathit{enat})$$

introduces a type that models the natural numbers Z, $\mathsf{S}(\mathsf{Z})$, $\mathsf{S}(\mathsf{S}(\mathsf{Z}))$, …, in Peano notation but extended with an infinite value $\infty = \mathsf{S}(\mathsf{S}(\mathsf{S}(\ldots)))$. The equation $\mathsf{S}(\infty) \approx \infty$ holds as expected, because both sides expand to the infinite term $\mathsf{S}(\mathsf{S}(\mathsf{S}(\ldots)))$, which uniquely identifies ∞. Compared with the conventional definition **datatype** $\mathit{enat} = \mathsf{Z} \mid \mathsf{S}(\mathit{enat}) \mid \mathsf{Infty}$, the codatatype avoids one case by unifying the infinite and finite nonzero cases.

Datatypes and codatatypes are an integral part of many proof assistants, including Agda, Coq, Isabelle, Matita, and PVS. In recent years, datatypes have emerged in a few automatic theorem provers as well. The SMT-LIB format, implemented by most SMT solvers, has been extended with a syntax for datatypes. In this paper, we introduce a unified decision procedure for universal problems involving datatypes and codatatypes

in combination (Section 3). The procedure is described abstractly as a calculus and is composable via the Nelson–Oppen method [18]. It generalizes the procedure by Barrett et al. [2], which covers only datatypes. Detailed proofs are included in a report [20].

Datatypes and codatatypes share many properties, so it makes sense to consider them together. There are, however, at least three important differences. First, *codatatypes need not be well-founded.* For example, the type **codatatype** $\mathit{stream}_\tau = \mathsf{SCons}(\tau, \mathit{stream}_\tau)$ of infinite sequences or streams over an element type τ is allowed, even though it has no base case. Second, *a uniqueness rule takes the place of the acyclicity rule of datatypes.* Cyclic constraints such as $x \approx \mathsf{S}(x)$ are unsatisfiable for datatypes, thanks to an acyclicity rule, but satisfiable for codatatypes. For the latter, a uniqueness rule ensures that two values having the same infinite expansion must be equal; from $x \approx \mathsf{S}(x)$ and $y \approx \mathsf{S}(y)$, it deduces $x \approx y$. These two rules are needed to ensure completeness (solution soundness) on universal problems. They cannot be expressed as finite axiomatizations, so they naturally belong in a decision procedure. Third, *it must be possible to express cyclic* (*regular*) *values as closed terms and to enumerate them.* This is necessary both for finite model finding [22] and for theory combinations. The μ-binder notation associates a name with a subterm; it is used to represent cyclic values in the generated models. For example, the μ-term $\mathsf{SCons}(1, \mu s.\ \mathsf{SCons}(0, \mathsf{SCons}(9, s)))$ stands for the lasso-shaped sequence $1,0,9,0,9,0,9,\ldots$.

Our procedure is implemented in the SMT solver CVC4 [1] as a combination of rewriting and a theory solver (Section 4). It consists of about 2000 lines of C++ code, most of which are shared between datatypes and codatatypes. The code is integrated in the development version of the solver and is expected to be part of the CVC4 1.5 release. An evaluation on problems generated from Isabelle theories using the Sledgehammer tool [3] demonstrates the usefulness of the approach (Section 5).

Barrett et al. [2] provide a good account of related work on datatypes as of 2007, in addition to describing their implementation in CVC3. Since then, datatypes have been added not only to CVC4 (a complete rewrite of CVC3) but also to the SMT solver Z3 [17] and a SPASS-like superposition prover [27]. Closely related are the automatic structural induction in both kinds of provers [9, 21], the (co)datatype and (co)induction support in Dafny [12], and the (semi-)decision procedures for datatypes implemented in Leon [26] and RADA [19]. Datatypes are supported by the higher-order model finder Refute [28]. Its successor, Nitpick [4], can also generate models involving cyclic codatatype values. Cyclic values have been studied extensively under the heading of regular or rational trees—see Carayol and Morvan [5] and Djellou et al. [6] for recent work. The μ-notation is inspired by the μ-calculus [11].

Conventions. Our setting is a monomorphic (or many-sorted) first-order logic. A signature $\Sigma = (\mathcal{Y}, \mathcal{F})$ consists of a set of types $\mathcal{Y}$ and a set of function symbols $\mathcal{F}$. Types are atomic sorts and interpreted as nonempty domains. The set $\mathcal{Y}$ must contain a distinguished type *bool* interpreted as the set of truth values. Names starting with an uppercase letter are reserved for constructors. With each function symbol f is associated a list of argument types $\tau_1, \ldots, \tau_n$ (for $n \geq 0$) and a return type τ, written $\mathsf{f} : \tau_1 \times \cdots \times \tau_n \to \tau$. The set $\mathcal{F}$ must at least contain $\mathsf{true}, \mathsf{false} : \mathit{bool}$, interpreted as truth values. The only predicate is equality ($\approx$). The notation t^τ stands for a term t of type τ. When applied to terms, the symbol $=$ denotes syntactic equality.

2 (Co)datatypes

We fix a signature $\Sigma = (\mathcal{Y}, \mathcal{F})$. The types are partitioned into $\mathcal{Y} = \mathcal{Y}_{\text{dt}} \uplus \mathcal{Y}_{\text{codt}} \uplus \mathcal{Y}_{\text{ord}}$, where $\mathcal{Y}_{\text{dt}}$ are the *datatypes*, $\mathcal{Y}_{\text{codt}}$ are the *codatatypes*, and $\mathcal{Y}_{\text{ord}}$ are the *ordinary types* (which can be interpreted or not). The function symbols are partitioned into $\mathcal{F} = \mathcal{F}_{\text{ctr}} \uplus \mathcal{F}_{\text{sel}}$, where $\mathcal{F}_{\text{ctr}}$ are the *constructors* and $\mathcal{F}_{\text{sel}}$ are the *selectors*. There is no need to consider further function symbols because they can be abstracted away as variables when combining theories. Σ-terms are standard first-order terms over Σ, without μ-binders.

In an SMT problem, the signature is typically given by specifying first the uninterpreted types in any order, then the (co)datatypes with their constructors and selectors in groups of mutually (co)recursive groups of (co)datatypes, and finally any other function symbols. Each (co)datatype specification consists of ℓ mutually recursive types that are either all datatypes or all codatatypes. Nested (co)recursion and datatype–codatatype mixtures fall outside this fragment.

Each (co)datatype δ is equipped with $m \geq 1$ constructors, and each constructor for δ takes zero or more arguments and returns a δ value. The argument types must be either ordinary, among the already known (co)datatypes, or among the (co)datatypes being introduced. To every argument corresponds a selector. The names for the (co)datatypes, the constructors, and the selectors must be fresh. Schematically:

$$\begin{aligned}(\textbf{co})\textbf{datatype}\ \delta_1 &= \mathsf{C}_{11}(\left[\mathsf{s}^1_{11}{:}\right]\tau^1_{11}, \ldots, \left[\mathsf{s}^{n_{11}}_{11}{:}\right]\tau^{n_{11}}_{11}) \mid \cdots \mid \mathsf{C}_{1m_1}(\ldots)\\ &\vdots\\ \textbf{and}\ \delta_\ell &= \mathsf{C}_{\ell 1}(\ldots) \mid \cdots \mid \mathsf{C}_{\ell m_\ell}(\ldots)\end{aligned}$$

with $\mathsf{C}_{ij} : \tau^1_{ij} \times \cdots \times \tau^{n_{ij}}_{ij} \to \delta_i$ and $\mathsf{s}^k_{ij} : \delta_i \to \tau^k_{ij}$. Defaults are assumed for the selector names if they are omitted. The δ constructors and selectors are denoted by $\mathcal{F}^\delta_{\text{ctr}}$ and $\mathcal{F}^\delta_{\text{sel}}$. For types with several constructors, it is useful to provide discriminators $\mathsf{d}_{ij} : \delta_i \to \mathit{bool}$. Instead of extending $\mathcal{F}$, we let $\mathsf{d}_{ij}(t)$ be an abbreviation for $t \approx \mathsf{C}_{ij}\big(\mathsf{s}^1_{ij}(t), \ldots, \mathsf{s}^{n_{ij}}_{ij}(t)\big)$.

A type δ depends on another type ε if ε is the type of an argument to one of δ's constructors. Semantically, a set of types is *mutually* (*co*)*recursive* if and only if the associated dependency graph is strongly connected. A type is (*co*)*recursive* if it belongs to such a set of types. Non(co)recursive type specifications such as **datatype** $\mathit{option}_\tau = \mathsf{None} \mid \mathsf{Some}(\tau)$ are permitted.

One way to characterize datatypes is as the initial model of the selector–constructor equations [2]. A related semantic view of datatypes is as initial algebras. Codatatypes are then defined dually as final coalgebras [24]. The datatypes are generated by their constructors, whereas the codatatypes are viewed through their selectors.

Datatypes and codatatypes share many basic properties:

$$\begin{aligned}\text{Distinctness:}\quad & \mathsf{C}_{ij}(\bar{x}) \not\approx \mathsf{C}_{ij'}(\bar{y}) \quad \text{if } j \neq j'\\ \text{Injectivity:}\quad & \mathsf{C}_{ij}(x_1, \ldots, x_{n_{ij}}) \approx \mathsf{C}_{ij}(y_1, \ldots, y_{n_{ij}}) \longrightarrow x_k \approx y_k\\ \text{Exhaustiveness:}\quad & \mathsf{d}_{i1}(x) \vee \cdots \vee \mathsf{d}_{im_i}(x)\\ \text{Selection:}\quad & \mathsf{s}^k_{ij}(\mathsf{C}_{ij}(x_1, \ldots, x_{n_{ij}})) \approx x_k\end{aligned}$$

Datatypes are additionally characterized by an induction principle. The principle ensures that the interpretation of datatypes is standard. For the natural numbers constructed from Z and S, induction prohibits models that contain cyclic values—e.g., an n such that $n \approx \mathsf{S}(n)$—or even infinite acyclic values $\mathsf{S}(\mathsf{S}(\ldots))$.

For codatatypes, the dual notion is called coinduction. This axiom encodes a form of extensionality: Two values that yield the same observations must be equal, where the observations are made through selectors and discriminators. In addition, codatatypes are guaranteed to contain all values corresponding to infinite ground constructor terms.

Given a signature Σ, $\mathcal{DC}$ refers to the *theory of datatypes and codatatypes*, which defines a class of Σ-interpretations $\mathscr{J}$, namely the ones that satisfy the properties mentioned in this section, including (co)induction. The interpretations in $\mathscr{J}$ share the same interpretation for constructor terms and correctly applied selector terms (up to isomorphism) but may differ on variables and wrongly applied selector terms. A formula is *$\mathcal{DC}$-satisfiable* if there exists an interpretation in $\mathscr{J}$ that satisfies it. For deciding universal formulas, induction can be replaced by the acyclicity axiom schema, which states that constructor terms cannot be equal to any of their proper subterms [2]. Dually, coinduction can be replaced by the uniqueness schema, which asserts that codatatype values are fully characterized by their expansion [24, Theorem 8.1, $2 \Leftrightarrow 5$].

Some codatatypes are so degenerate as to be finite even though they have infinite values. A simple example is **codatatype** $a = \mathsf{A}(a)$, whose unique value is $\mu a.\, \mathsf{A}(a)$. Other specimens are $\mathit{stream}_{\mathit{unit}}$ and **codatatype** $b = \mathsf{B}(b, c, b, \mathit{unit})$ **and** $c = \mathsf{C}(a, b, c)$, where *unit* is a datatype with the single constructor $\mathsf{Unity} : \mathit{unit}$. We call such types *corecursive singletons*. For the decision procedure, it will be crucial to detect these. A type may also be a corecursive singleton only in some models. If the example above is altered to make *unit* an uninterpreted type, b and c will be singletons precisely when *unit* is interpreted as a singleton. Fortunately, it is easy to characterize this degenerate case.

Lemma 1. *Let δ be a corecursive codatatype. For any interpretation in $\mathscr{J}$, the domain interpreting δ is either infinite or a singleton. In the latter case, δ necessarily has a single constructor, whose arguments have types that are interpreted as singleton domains.*

3 The Decision Procedure

Given a fixed signature Σ, the decision procedure for the universal theory of (co)datatypes determines the $\mathcal{DC}$-satisfiability of finite sets E of literals: equalities and disequalities between Σ-terms, whose variables are interpreted existentially. The procedure is formulated as a tableau-like calculus. Proving a universal quantifier-free conjecture is reduced to showing that its negation is unsatisfiable. The presentation is inspired by Barrett et al. [2] but higher-level, using unoriented equations instead of oriented ones.

To simplify the presentation, we make a few assumptions about Σ. First, all codatatypes are corecursive. This is reasonable because noncorecursive codatatypes can be seen as nonrecursive datatypes. Second, all ordinary types have infinite cardinality. Without quantifiers, the constraints E cannot entail an upper bound on the cardinality of any uninterpreted type, so it is safe to consider these types infinite. As for ordinary types interpreted finitely by other theories (e.g., bit vectors), each interpreted type having finite cardinality n can be viewed as a datatype with n nullary constructors [2].

A derivation rule can be applied to E if the preconditions are met. The conclusion either specifies equalities to be added to E or is $\bot$ (contradiction). One rule has multiple conclusions, denoting branching. An application of a rule is *redundant* if one of its non-$\bot$ conclusions leaves E unchanged. A *derivation tree* is a tree whose nodes are finite

sets of equalities, such that child nodes are obtained by a nonredundant application of a derivation rule to the parent. A derivation tree is *closed* if all of its leaf nodes are $\bot$. A node is *saturated* if no nonredundant instance of a rule can be applied to it.

The calculus consists of three sets of rules, given in Figures 1 to 3, corresponding to three phases. The first phase computes the bidirectional closure of E. The second phase makes inferences based on acyclicity (for datatypes) and uniqueness (for codatatypes). The third phase performs case distinctions on constructors for various terms occurring in E. The rules belonging to a phase have priority over those of subsequent phases. The rules are applied until the derivation tree is closed or all leaf nodes are saturated.

Phase 1: Computing the Bidirectional Closure (Figure 1). In conjunction with Refl, Sym, and Trans, the Cong rule computes the congruence (upward) closure, whereas the Inject and Clash rules compute the unification (downward) closure. For unification, equalities are inferred based on the injectivity of constructors by Inject, and failures to unify equated terms are recognized by Clash. The Conflict rule recognizes when an equality and its negation both occur in E, in which case E has no model.

Let $\mathcal{T}(E)$ denote the set of Σ-terms occurring in E. At the end of the first phase, E induces an equivalence relation over $\mathcal{T}(E)$ such that two terms t and u are equivalent if and only if $t \approx u \in E$. Thus, we can regard E as a set of equivalence classes of terms. For a term $t \in \mathcal{T}(E)$, we write $[t]$ to denote the equivalence class of t in E.

Phase 2: Applying Acyclicity and Uniqueness (Figure 2). We describe the rules in this phase in terms of a mapping $\mathcal{A}$ that associates with each equivalence class a μ-term as its representative.

Formally, *μ-terms* are defined recursively as being either a variable x or an applied constructor $\mu x.\, \mathsf{C}(\bar{t})$ for some $\mathsf{C} \in \mathcal{F}_{\text{ctr}}$ and μ-terms $\bar{t}$ of the expected types. The variable x need not occur free in the μ-binder's body, in which case the binder can be omitted. $\mathrm{FV}(t)$ denotes the set of free variables occurring in the μ-term t. A μ-term is *closed* if it contains no free variables. It is *cyclic* if it contains a bound variable. The *α-equivalence* relation $t =_\alpha u$ indicates that the μ-terms t and u are syntactically equivalent for some capture-avoiding renaming of μ-bound variables. Two μ-terms can denote the same value despite being α-disequivalent—e.g., $\mu x.\, \mathsf{S}(x) \neq_\alpha \mu y.\, \mathsf{S}(\mathsf{S}(y))$.

The μ-term $\mathcal{A}[t^\tau]$ describes a class of τ values that t and other members of t's equivalence class can take in models of E. When τ is a datatype, a cyclic μ-term describes an infeasible class of values.

The mapping $\mathcal{A}$ is defined as follows. With each equivalence class $[u^\tau]$, we associate a fresh variable $\widetilde{u}^\tau$ and set $\mathcal{A}[u] := \widetilde{u}$, that is to say there are initially no constraints on the values for any equivalence class $[u]$. The mapping $\mathcal{A}$ is refined by applying the following unfolding rule exhaustively:

$$\frac{\widetilde{u} \in \mathrm{FV}(\mathcal{A}) \quad \mathsf{C}(t_1, \ldots, t_n) \in [u] \quad \mathsf{C} \in \mathcal{F}_{\text{ctr}}}{\mathcal{A} := \mathcal{A}[\widetilde{u} \mapsto \mu\widetilde{u}.\, \mathsf{C}(\widetilde{t_1}, \ldots, \widetilde{t_n})]}$$

$\mathrm{FV}(\mathcal{A})$ denotes the set of free variables occurring in $\mathcal{A}$'s range, and $\mathcal{A}[x \mapsto t]$ denotes the *variable-capturing* substitution of t for x in $\mathcal{A}$'s range. It is easy to see that the height of terms produced as a result of the unfolding is bounded by the number of equivalence classes of E, and thus the construction of $\mathcal{A}$ will terminate.

$$\frac{t \in \mathcal{T}(E)}{E := E,\, t \approx t}\ \textsf{Refl} \qquad \frac{t \approx u \in E}{E := E,\, u \approx t}\ \textsf{Sym} \qquad \frac{s \approx t,\, t \approx u \in E}{E := E,\, s \approx u}\ \textsf{Trans}$$

$$\frac{\bar{t} \approx \bar{u} \in E \quad \mathsf{f}(\bar{t}), \mathsf{f}(\bar{u}) \in \mathcal{T}(E)}{E := E,\, \mathsf{f}(\bar{t}) \approx \mathsf{f}(\bar{u})}\ \textsf{Cong} \qquad \frac{t \approx u,\, t \not\approx u \in E}{\bot}\ \textsf{Conflict}$$

$$\frac{\mathsf{C}(\bar{t}) \approx \mathsf{C}(\bar{u}) \in E}{E := E,\, \bar{t} \approx \bar{u}}\ \textsf{Inject} \qquad \frac{\mathsf{C}(\bar{t}) \approx \mathsf{D}(\bar{u}) \in E \quad \mathsf{C} \neq \mathsf{D}}{\bot}\ \textsf{Clash}$$

Figure 1. Derivation rules for bidirectional closure

$$\frac{\delta \in \mathcal{Y}_{\text{dt}} \quad \mathcal{A}[t^\delta] = \mu x.\, u \quad x \in \text{FV}(u)}{\bot}\ \textsf{Acyclic} \qquad \frac{\delta \in \mathcal{Y}_{\text{codt}} \quad \mathcal{A}[t^\delta] =_\alpha \mathcal{A}[u^\delta]}{E := E,\, t \approx u}\ \textsf{Unique}$$

Figure 2. Derivation rules for acyclicity and uniqueness

$$\frac{\begin{array}{c} t^\delta \in \mathcal{T}(E) \quad \mathcal{F}^\delta_{\text{ctr}} = \{\mathsf{C}_1, \ldots, \mathsf{C}_m\} \\ \big(\mathsf{s}(t) \in \mathcal{T}(E) \text{ and } \mathsf{s} \in \mathcal{F}^\delta_{\text{sel}}\big) \text{ or } \big(\delta \in \mathcal{Y}_{\text{dt}} \text{ and } \delta \text{ is finite}\big) \end{array}}{E := E,\, t \approx \mathsf{C}_1\big(\mathsf{s}^1_1(t), \ldots, \mathsf{s}^{n_1}_1(t)\big) \quad \cdots \quad E := E,\, t \approx \mathsf{C}_m\big(\mathsf{s}^1_m(t), \ldots, \mathsf{s}^{n_m}_m(t)\big)}\ \textsf{Split}$$

$$\frac{t^\delta, u^\delta \in \mathcal{T}(E) \quad \delta \in \mathcal{Y}_{\text{codt}} \quad \delta \text{ is a singleton}}{E := E,\, t \approx u}\ \textsf{Single}$$

Figure 3. Derivation rules for branching

Example 1. Suppose that E contains four distinct equivalence classes $[w]$, $[x]$, $[y]$, and $[z]$ such that $\mathsf{C}(w,y) \in [x]$ and $\mathsf{C}(z,x) \in [y]$ for some $\mathsf{C} \in \mathcal{F}_{\text{ctr}}$. A possible sequence of unfolding steps is given below, omitting trivial entries such as $[w] \mapsto \widetilde{w}$.

1. Unfold $\widetilde{x}$: $\mathcal{A} = \{[x] \mapsto \mu\widetilde{x}.\, \mathsf{C}(\widetilde{w}, \widetilde{y})\}$
2. Unfold $\widetilde{y}$: $\mathcal{A} = \{[x] \mapsto \mu\widetilde{x}.\, \mathsf{C}(\widetilde{w}, \mu\widetilde{y}.\, \mathsf{C}(\widetilde{z}, \widetilde{x})),\ [y] \mapsto \mu\widetilde{y}.\, \mathsf{C}(\widetilde{z}, \widetilde{x})\}$
3. Unfold $\widetilde{x}$: $\mathcal{A} = \{[x] \mapsto \mu\widetilde{x}.\, \mathsf{C}(\widetilde{w}, \mu\widetilde{y}.\, \mathsf{C}(\widetilde{z}, \widetilde{x})),\ [y] \mapsto \mu\widetilde{y}.\, \mathsf{C}(\widetilde{z}, \mu\widetilde{x}.\, \mathsf{C}(\widetilde{w}, \widetilde{y}))\}$

The resulting $\mathcal{A}$ indicates that the values for x and y in models of E must be of the forms $\mathsf{C}(\widetilde{w}, \mathsf{C}(\widetilde{z}, \mathsf{C}(\widetilde{w}, \mathsf{C}(\widetilde{z}, \ldots))))$ and $\mathsf{C}(\widetilde{z}, \mathsf{C}(\widetilde{w}, \mathsf{C}(\widetilde{z}, \mathsf{C}(\widetilde{w}, \ldots))))$, respectively. ∎

Given the mapping $\mathcal{A}$, the Acyclic and Unique rules work as follows. For acyclicity, if $[t]$ is a datatype equivalence class whose values $\mathcal{A}[t] = \mu x.\, u$ are cyclic (as expressed by $x \in \text{FV}(u)$), then E is $\mathcal{DC}$-unsatisfiable. For uniqueness, if $[t]$, $[u]$ are two codatatype equivalence classes whose values $\mathcal{A}[t]$, $\mathcal{A}[u]$ are α-equivalent, then $t \approx u$. Comparison for α-equivalence may seem too restrictive, since $\mu x.\, \mathsf{S}(x)$ and $\mu y.\, \mathsf{S}(\mathsf{S}(y))$ specify the same value despite being α-disequivalent, but the rule will make progress by discovering that the subterm $\mathsf{S}(y)$ of $\mu y.\, \mathsf{S}(\mathsf{S}(y))$ must be equal to the entire term.

Example 2. Let $E = \{x \approx \mathsf{S}(x),\ y \approx \mathsf{S}(\mathsf{S}(y))\}$. After phase 1, the equivalence classes are $\{x, \mathsf{S}(x)\}$, $\{y, \mathsf{S}(\mathsf{S}(y))\}$, and $\{\mathsf{S}(y)\}$. Constructing $\mathcal{A}$ yields

$$\mathcal{A}[x] = \mu\widetilde{x}.\, \mathsf{S}(\widetilde{x}) \qquad \mathcal{A}[y] = \mu\widetilde{y}.\, \mathsf{S}(\mu\widetilde{\mathsf{S}(y)}.\, \mathsf{S}(\widetilde{y})) \qquad \mathcal{A}[\mathsf{S}(y)] = \mu\widetilde{\mathsf{S}(y)}.\, \mathsf{S}(\mu\widetilde{y}.\, \mathsf{S}(\widetilde{\mathsf{S}(y)}))$$

Since $\mathcal{A}[y] =_\alpha \mathcal{A}[\mathsf{S}(y)]$, the Unique rule applies to derive $y \approx \mathsf{S}(y)$. At this point, phase 1 is activated again, yielding $\{x, \mathsf{S}(x)\}$ and $\{y, \mathsf{S}(y), \mathsf{S}(\mathsf{S}(y))\}$. The mapping $\mathcal{A}$ is updated accordingly: $\mathcal{A}[y] = \mu\widetilde{y}.\, \mathsf{S}(\widetilde{y})$. Since $\mathcal{A}[x] =_\alpha \mathcal{A}[y]$, Unique can finally derive $x \approx y$. ■

Phase 3: Branching (Figure 3). If a selector is applied to a term t, or if t's type is a finite datatype, t's equivalence class must contain a δ constructor term. This is enforced in the third phase by the Split rule. Another rule, Single, focuses on the degenerate case where two terms have the same corecursive singleton type and are therefore equal. Both Split's finiteness assumption and Single's singleton constraint can be evaluated statically based on a recursive computation of the cardinalities of the constructors' argument types.

Correctness. Correctness means that if there exists a closed derivation tree with root node E, then E is $\mathcal{DC}$-unsatisfiable; and if there exists a derivation tree with root node E that contains a saturated node, then E is $\mathcal{DC}$-satisfiable.

Theorem 2 (Termination). *All derivation trees are finite.*

Proof. Consider a derivation tree with root node E. Let $D \subseteq \mathcal{T}(E)$ be the set of terms whose types are finite datatypes, and let $S \subseteq \mathcal{T}(E)$ be the set of terms occurring as arguments to selectors. For each term $t \in D$, let $S_t^0 = \{t\}$ and $S_t^{i+1} = S_t^i \cup \{\mathsf{s}(u) \mid u^\delta \in S_t^i,\ \delta \in \mathcal{Y}_{\mathrm{dt}},\ |\delta| \text{ is finite}, \mathsf{s} \in \mathcal{F}_{\mathrm{sel}}^\delta\}$, and let S_t^∞ be the limit of this sequence. This is a finite set for each t, because all chains of selectors applied to t are finite. Let S^∞ be the union of all sets S_t^∞ where $t \in D$, and let $\mathcal{T}^\infty(E)$ be the set of subterms of $E \cup \{\mathsf{C}_j\big(\mathsf{s}_j^1(t), \ldots, \mathsf{s}_j^{n_j}(t)\big) \mid t^\delta \in S \cup S^\infty, \mathsf{C}_j \in \mathcal{F}_{\mathrm{ctr}}^\delta\}$. In a derivation tree with root node E, it can be shown by induction on the rules of the calculus that each non-root node F is such that $\mathcal{T}(F) \subseteq \mathcal{T}^\infty(E)$, and hence contains an equality between two terms from $\mathcal{T}^\infty(E)$ not occurring in its parent node. Thus, the depth of a branch in a derivation tree with root node E is at most $|\mathcal{T}^\infty(E)|^2$, which is finite since $\mathcal{T}^\infty(E)$ is finite. □

Theorem 3 (Refutation Soundness). *If there exists a closed derivation tree with root node E, then E is $\mathcal{DC}$-unsatisfiable.*

Proof. The proof is by structural induction on the derivation tree with root node E. If the tree is an application of Conflict, Clash, or Acyclic, then E is $\mathcal{DC}$-unsatisfiable. For Conflict, this is a consequence of equality reasoning. For Clash, this is a consequence of distinctness. For Acyclic, the construction of $\mathcal{A}$ indicates that the class of values that term t can take in models of E is infeasible. If the child nodes of E are closed derivation trees whose root nodes are the result of applying Split on t^δ, by the induction hypothesis $E \cup t \approx \mathsf{C}_j\big(\mathsf{s}_j^1(t), \ldots, \mathsf{s}_j^{n_j}(t)\big)$ is $\mathcal{DC}$-unsatisfiable for each $\mathsf{C}_j \in \mathcal{F}_{\mathrm{ctr}}^\delta$. Since by exhaustiveness, all models of $\mathcal{DC}$ entail exactly one $t \approx \mathsf{C}_j\big(\mathsf{s}_j^1(t), \ldots, \mathsf{s}_j^{n_j}(t)\big)$, E is $\mathcal{DC}$-unsatisfiable. Otherwise, the child node of E is a closed derivation tree whose root node $E \cup t \approx u$ is obtained by applying one of the rules Refl, Sym, Trans, Cong, Inject, Unique, or Single. In all these cases, $E \vDash_{\mathcal{DC}} t \approx u$. For Refl, Sym, Trans, Cong, this is a consequence of equality reasoning. For Inject, this is a consequence of injectivity. For Unique, the construction of $\mathcal{A}$ indicates that the values of t and u are equivalent in all models of E. For Single, t and u must have the same value since the cardinality of their type is one. By the induction hypothesis, $E \cup t \approx u$ is $\mathcal{DC}$-unsatisfiable and thus E is $\mathcal{DC}$-unsatisfiable. □

It remains to show the converse of the previous theorem: If a derivation tree with root node E contains a saturated node, then E is $\mathcal{DC}$-satisfiable. The proof relies on a specific interpretation $\mathcal{I}$ that satisfies E.

First, we define the set of interpretations of the theory $\mathcal{DC}$, which requires custom terminology concerning μ-terms. Given a μ-term t with subterm u, the *expansion of u with respect to t* is the μ-term $\langle u \rangle_t^{\emptyset}$, abbreviated to $\langle u \rangle_t$, as returned by the function

$$\langle x \rangle_t^B = \begin{cases} x & \text{if } x \in B \\ \mu x.\, \mathsf{C}\big(\langle \bar{u} \rangle_t^{B \uplus \{x\}}\big) & \text{if } \mu x.\, \mathsf{C}(\bar{u}) \text{ binds this occurrence of } x \notin B \text{ in } t \end{cases}$$

$$\langle \mu x.\, \mathsf{C}(\bar{u}) \rangle_t^B = \begin{cases} x & \text{if } x \in B \\ \mu x.\, \mathsf{C}\big(\langle \bar{u} \rangle_t^{B \uplus \{x\}}\big) & \text{otherwise} \end{cases}$$

The recursion will eventually terminate because each recursion adds one bound variable to B and there are finitely many distinct bound variables in a μ-term. Intuitively, the expansion of a subterm is a stand-alone μ-term that denotes the same value as the original subterm—e.g., $\langle \mu y.\, \mathsf{D}(x) \rangle_{\mu x.\, \mathsf{C}(\mu y.\, \mathsf{D}(x))} = \mu y.\, \mathsf{D}(\mu x.\, \mathsf{C}(y))$.

The μ-term u is a *self-similar subterm* of t if u is a proper subterm of t, t and u are of the forms $\mu x.\, \mathsf{C}(t_1, \ldots, t_n)$ and $\mu y.\, \mathsf{C}(u_1, \ldots, u_n)$, and $\langle t_k \rangle_t =_\alpha \langle u_k \rangle_t$ for all k. The μ-term t is *normal* if it does not contain self-similar subterms and all of its proper subterms are also normal. Thus, $t = \mu x.\, \mathsf{C}(\mu y.\, \mathsf{C}(y))$ is not normal because $\mu y.\, \mathsf{C}(y)$ is a self-similar subterm of t. Their arguments have the same expansion with respect to t: $\langle \mu y.\, \mathsf{C}(y) \rangle_t = \mu y.\, \mathsf{C}\big(\langle y \rangle_t^{\{y\}}\big) = \mu y.\, \mathsf{C}(y)$ is α-equivalent to $\langle y \rangle_t = \mu y.\, \mathsf{C}\big(\langle y \rangle_t^{\{y\}}\big) = \mu y.\, \mathsf{C}(y)$. The term $u = \mu x.\, \mathsf{C}(\mu y.\, \mathsf{C}(x))$ is also not normal, since $\mu y.\, \mathsf{C}(x)$ is a self-similar subterm of u, noting that $\langle \mu y.\, \mathsf{C}(x) \rangle_u = \mu y.\, \mathsf{C}\big(\langle x \rangle_u^{\{y\}}\big) = \mu y.\, \mathsf{C}\big(\langle \mu x.\, \mathsf{C}(\mu y.\, \mathsf{C}(x)) \rangle_u^{\{y\}}\big) = \mu y.\, \mathsf{C}\big(\mu x.\, \mathsf{C}(\langle \mu y.\, \mathsf{C}(x) \rangle_u^{\{x,y\}})\big) = \mu y.\, \mathsf{C}(\mu x.\, \mathsf{C}(y))$ is α-equivalent to $\langle x \rangle_u = u$.

For any μ-term t of the form $\mu x.\, \mathsf{C}(\bar{u})$, its *normal form* $\lfloor t \rfloor$ is obtained by replacing all of the self-similar subterms of t with x and by recursively normalizing the other subterms. For variables, $\lfloor x \rfloor = x$. Thus, $\lfloor \mu x.\, \mathsf{C}(\mu y.\, \mathsf{C}(x)) \rfloor = \mu x.\, \mathsf{C}(x)$.

We now define the class of interpretations for $\mathcal{DC}$. $\mathcal{I}(\tau)$ denotes the interpretation type τ in $\mathcal{I}$—that is, a nonempty set of domain elements for that type. $\mathcal{I}(\mathsf{f})$ denotes the interpretation of a function f in $\mathcal{I}$. If $\mathsf{f} : \tau_1 \times \cdots \times \tau_n \to \tau$, then $\mathcal{I}(\mathsf{f})$ is a total function from $\mathcal{I}(\tau_1) \times \cdots \times \mathcal{I}(\tau_n)$ to $\mathcal{I}(\tau)$. All types are interpreted as sets of μ-terms, but only values of types in $\mathcal{Y}_{\text{codt}}$ may contain cycles.

Definition 4 (Normal Interpretation). An interpretation $\mathcal{I}$ is *normal* if these conditions are met:

1. For each type τ, $\mathcal{I}(\tau)$ includes a maximal set of closed normal μ-terms of that type that are unique up to α-equivalence and acyclic if $\tau \notin \mathcal{Y}_{\text{codt}}$.
2. For each constructor term $\mathsf{C}(\bar{t})$ of type τ, $\mathcal{I}(\mathsf{C})(\mathcal{I}(\bar{t}))$ is the value in $\mathcal{I}(\tau)$ that is α-equivalent to $\lfloor \mu x.\, \mathsf{C}(\mathcal{I}(\bar{t})) \rfloor$, where x is fresh.
3. For each selector term $\mathsf{s}_j^k(t)$ of type τ, if $\mathcal{I}(t)$ is $\mu x.\, \mathsf{C}_j(\bar{u})$, then $\mathcal{I}(\mathsf{s}_j^k)\big(\mathcal{I}(t)\big)$ is the value in $\mathcal{I}(\tau)$ that is α-equivalent to $\langle u_k \rangle_{\mathcal{I}(t)}$.

Not all normal interpretations are models of codatatypes, because models must contain all possible infinite terms, not only cyclic ones. However, acyclic infinite values are uninteresting to us, and for quantifier-free formulas it is trivial to extend any normal interpretation with extra domain elements to obtain a genuine model if desired.

When constructing a model $\mathcal{J}$ of E, it remains only to specify how $\mathcal{J}$ interprets wrongly applied selector terms and variables. For the latter, this will be based on the mapping $\mathcal{A}$ computed in phase 2 of the calculus.

First, we need the following definitions. We write $t =_\alpha^x u$ if μ-terms t and u are syntactically equivalent for some renaming that avoids capturing any variable other than x. For example, $\mu x.\, \mathsf{D}(x) =_\alpha^y \mu x.\, \mathsf{D}(y)$ (by renaming y to x), $\mu x.\, \mathsf{C}(x,x) =_\alpha^x \mu y.\, \mathsf{C}(x,y)$, and $\mu x.\, \mathsf{C}(z,x) =_\alpha^z \mu y.\, \mathsf{C}(z,y)$, but $\mu x.\, \mathsf{D}(x) \neq_\alpha^x \mu x.\, \mathsf{D}(y)$ and $\mu x.\, \mathsf{C}(x,x) \neq_\alpha^y \mu y.\, \mathsf{C}(x,y)$. For a variable x^τ and a normal interpretation $\mathcal{J}$, we let $\mathcal{V}_\mathcal{J}^x(\mathcal{A})$ denote the set consisting of all values $v \in \mathcal{J}(\tau)$ such that $v =_\alpha^x \langle u \rangle_t$ for some subterm u of a term t occurring in the range of $\mathcal{A}$. This set describes shapes of terms to avoid when assigning a μ-term to x.

The *completion* $\mathcal{A}^\star$ of $\mathcal{A}$ for a normal interpretation $\mathcal{J}$ assigns values from $\mathcal{J}$ to unassigned variables in the domain of $\mathcal{A}$. We construct $\mathcal{A}^\star$ by initially setting $\mathcal{A}^\star := \lfloor \mathcal{A} \rfloor$ and by exhaustively applying the following rule:

$$\frac{\widetilde{x}^\tau \in \mathrm{FV}(\mathcal{A}^\star) \quad \mu\widetilde{x}.\, t =_\alpha v \quad v \in \mathcal{J}(\tau) \quad v \notin \mathcal{V}_\mathcal{J}^{\widetilde{x}}(\mathcal{A}^\star)}{\mathcal{A}^\star := \lfloor \mathcal{A}^\star[\widetilde{x} \mapsto \mu\widetilde{x}.\, t] \rfloor}$$

Given an unassigned variable in $\mathcal{A}^\star$, this rule assigns it a fresh value—one that does not occur in $\mathcal{V}_\mathcal{J}^{\widetilde{x}}(\mathcal{A}^\star)$ modulo α-equivalence—excluding not only existing terms in the range of $\mathcal{A}^\star$ but also terms that could emerge as a result of the update. Since this update removes one variable from $\mathrm{FV}(\mathcal{A}^\star)$ and does not add any variables to $\mathrm{FV}(\mathcal{A}^\star)$, the process eventually terminates. We normalize all terms in the range of $\mathcal{A}^\star$ at each step.

To ensure disequality literals are satisfied by an interpretation based on $\mathcal{A}^\star$, it suffices that $\mathcal{A}^\star$ is injective modulo α-equivalence. This invariant holds initially, and the last precondition in the above rule ensures that it is maintained. The set $\mathcal{V}_\mathcal{J}^{\widetilde{x}}(\mathcal{A}^\star)$ is an overapproximation of the values that, when assigned to $\widetilde{x}$, will cause values in the range of $\mathcal{A}^\star$ to become α-equivalent. For infinite codatatypes, it is always possible to find fresh values v because $\mathcal{V}_\mathcal{J}^{\widetilde{x}}(\mathcal{A}^\star)$ is a finite set.

Example 3. Let δ be a codatatype with the constructors $\mathsf{C}, \mathsf{D}, \mathsf{E} : \delta \to \delta$. Let E be the set $\{u \approx \mathsf{C}(z),\ v \approx \mathsf{D}(z),\ w \approx \mathsf{E}(y),\ x \approx \mathsf{C}(v),\ z \not\approx v\}$. After applying the calculus to saturation on E, the mapping $\mathcal{A}$ is as follows:

$$\begin{array}{lll} \mathcal{A}[u] = \mu\widetilde{u}.\, \mathsf{C}(\widetilde{z}) & \mathcal{A}[w] = \mu\widetilde{w}.\, \mathsf{E}(\widetilde{y}) & \mathcal{A}[y] = \widetilde{y} \\ \mathcal{A}[v] = \mu\widetilde{v}.\, \mathsf{D}(\widetilde{z}) & \mathcal{A}[x] = \mu\widetilde{x}.\, \mathsf{C}(\mu\widetilde{v}.\, \mathsf{D}(\widetilde{z})) & \mathcal{A}[z] = \widetilde{z} \end{array}$$

To construct a completion $\mathcal{A}^\star$, we must choose values for $\widetilde{y}$ and $\widetilde{z}$, which are free in $\mathcal{A}$. Modulo α-equivalence, $\mathcal{V}_\mathcal{J}^{\widetilde{z}}(\mathcal{A}) = \{\mu a.\, \mathsf{C}(a), \mu a.\, \mathsf{D}(a), \mu a.\, \mathsf{C}(\mathsf{D}(a)), \mathsf{C}(\mu a.\, \mathsf{D}(a))\}$. Now consider a normal interpretation $\mathcal{J}$ that evaluates variables in E based on $\mathcal{A}$: $\mathcal{J}(u) = \mathcal{A}[u]$, $\mathcal{J}(v) = \mathcal{A}[v]$, and so on. Assigning a value for $\mathcal{A}[z]$ that is α-equivalent to a value in $\mathcal{V}_\mathcal{J}^{\widetilde{z}}(\mathcal{A})$ may cause values in the range of $\mathcal{A}$ to become α-equivalent, which in turn may cause E to be falsified by $\mathcal{J}$. For example, assign $\mu\widetilde{z}.\, \mathsf{D}(\widetilde{z})$ for $\widetilde{z}$. After the substitution, $\mathcal{A}[v] = \mu\widetilde{v}.\, \mathsf{D}(\mu\widetilde{z}.\, \mathsf{D}(\widetilde{z}))$, which has normal form $\mu\widetilde{v}.\, \mathsf{D}(\widetilde{v})$, which is α-equivalent to $\mu\widetilde{z}.\, \mathsf{D}(\widetilde{z})$. However, this contradicts the disequality $z \not\approx v$ in E. On the other hand, if the value assigned to $\widetilde{z}$ is fresh, the values in the range of $\mathcal{A}$ remain α-disequivalent. We can assign a value such as $\mu\widetilde{z}.\, \mathsf{E}(\widetilde{z})$, $\mu\widetilde{z}.\, \mathsf{D}(\mathsf{C}(\widetilde{z}))$, or $\mu\widetilde{z}.\, \mathsf{C}(\mathsf{C}(\mathsf{D}(\widetilde{z})))$ to $\widetilde{z}$. ■

In the following lemma about $\mathcal{A}^\star$, $\mathrm{Var}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ x & \text{if } t \text{ is of the form } \mu x.\, u. \end{cases}$

Lemma 5. *If $\mathcal{A}$ is constructed for a saturated set E and $\mathcal{A}^\star$ is a completion of $\mathcal{A}$ for a normal interpretation $\mathcal{I}$, the following properties hold:*

(1) $\mathcal{A}^\star[x^\tau]$ *is α-equivalent to a value in* $\mathcal{I}(\tau)$.
(2) $\mathcal{A}^\star[x] = \langle t \rangle_{\mathcal{A}^\star[y]}$ *for all subterms t of* $\mathcal{A}^\star[y]$ *with* $\mathrm{Var}(t) = \widetilde{x}$.
(3) $\mathcal{A}^\star[x] =_\alpha \mathcal{A}^\star[y]$ *if and only if* $[x] = [y]$.

Intuitively, this lemma states three properties of $\mathcal{A}^\star$ that ensure a normal interpretation $\mathcal{I}$ can be constructed that satisfies E. Property (1) states that the values in the range of $\mathcal{A}^\star$ are α-equivalent to a value in normal interpretation. This means they are closed, normal, and acyclic when required. Property (2) states that the interpretation of all subterms in the range of $\mathcal{A}^\star$ depends on its associated variable only. In other words, the interpretation of a subterm t where $\mathrm{Var}(t) = \widetilde{x}$ is equal to $\mathcal{A}^\star[x]$, independently of the context. Property (3) states that $\mathcal{A}^\star$ is injective (modulo α-equivalence), which ensures that distinct values are assigned to distinct equivalence classes.

Theorem 6 (Solution Soundness). *If there exists a derivation tree with root node E containing a saturated node, then E is $\mathcal{DC}$-satisfiable.*

Proof. Let F be a saturated node in a derivation tree with root node E. We consider a normal interpretation $\mathcal{I}$ that interprets wrongly applied selectors based on equality information in F and that interprets the variables of F based on the completion $\mathcal{A}^\star$. For the variables, let $\mathcal{I}(x^\tau)$ be the value in $\mathcal{I}(\tau)$ that is α-equivalent with $\mathcal{A}^\star[x]$ for each variable $x \in \mathcal{T}(F)$, which by Lemma 5(1) is guaranteed to exist.

We first show that $\mathcal{I}$ satisfies all equalities $t_1 \approx t_2 \in F$. To achieve this, we show by structural induction on t^τ that $\mathcal{I}(t) =_\alpha \mathcal{A}^\star[t]$ for all terms $t \in \mathcal{T}(F)$, which implies $\mathcal{I} \models t_1 \approx t_2$ since $\mathcal{I}$ is normal.

If t is a variable, then $\mathcal{I}(t) =_\alpha \mathcal{A}^\star[t]$ by construction.

If t is a constructor term of the form $\mathsf{C}(u_1,\ldots,u_n)$, then $\mathcal{I}(t)$ is α-equivalent with $\lfloor \mu x.\, \mathsf{C}(\mathcal{I}(u_1),\ldots,\mathcal{I}(u_n)) \rfloor$ for some fresh x, which by the induction hypothesis is α-equivalent with $\lfloor \mu x.\, \mathsf{C}(\mathcal{A}^\star[u_1],\ldots,\mathcal{A}^\star[u_n]) \rfloor$. Call this term t'. Since Inject and Clash do not apply to F, by the construction of $\mathcal{A}^\star$ we have that $\mathcal{A}^\star[t]$ is a term of the form $\mu\widetilde{t}.\, \mathsf{C}(w_1,\ldots,w_n)$ where $\mathrm{Var}(w_i) = \widetilde{u_i}$ for each i. Thus by Lemma 5(2), $\langle w_i \rangle_{\mathcal{A}^\star[t]} = \mathcal{A}^\star[u_i]$. For each i, let $u_i{}'$ be the ith argument of t'. Clearly, $\langle u_i{}' \rangle_{t'} =_\alpha \mathcal{A}^\star[u_i]$. Thus $\langle u_i{}' \rangle_{t'} =_\alpha \langle w_i \rangle_{\mathcal{A}^\star[t]}$. Thus, $\mathcal{I}(t) =_\alpha t' =_\alpha \mathcal{A}^\star[t]$, and we have $\mathcal{I}(t) =_\alpha \mathcal{A}^\star[t]$.

If t is a selector term $\mathsf{s}^k_j(u)$, since Split does not apply to F, $[u]$ must contain a term of the form $\mathsf{C}_{j'}\big(\mathsf{s}^1_{j'}(u),\ldots,\mathsf{s}^n_{j'}(u)\big)$ for some j'. Since Inject and Clash are not applicable, by construction $\mathcal{A}^\star[u]$ must be of the form $\mu\widetilde{u}.\, \mathsf{C}_{j'}(w_1,\ldots,w_n)$, where $\mathrm{Var}(w_i) = \widetilde{\mathsf{s}^i_{j'}(u)}$ for each i, and thus by Lemma 5(2), $\langle w_i \rangle_{\mathcal{A}^\star[u]} = \mathcal{A}^\star[\mathsf{s}^i_{j'}(u)]$. If $j = j'$, then $\mathcal{I}(t)$ is α-equivalent with $\langle w_k \rangle_{\mathcal{A}^\star[u]}$, which is equal to $\mathcal{A}^\star[\mathsf{s}^k_j(u)] = \mathcal{A}^\star[t]$. If $j \neq j'$, since Cong does not apply, any term of the form $\mathsf{s}^k_j(u')$ not occurring in $[t]$ is such that $[u] \neq [u']$. By the induction hypothesis and Lemma 5(3), $\mathcal{I}(u) \neq \mathcal{I}(u')$ for all such u, u'. Thus, we may interpret $\mathcal{I}(\mathsf{s}^k_j)(\mathcal{I}(u))$ as the value in $\mathcal{I}(\tau)$ that is α-equivalent with $\mathcal{A}^\star[t]$.

We now show that all disequalities in F are satisfied by $\mathcal{I}$. Assume $t \not\approx u \in F$. Since Conflict does not apply, $t \approx u \notin F$ and thus $[t]$ and $[u]$ are distinct. Since $\mathcal{I}(t) =_\alpha \mathcal{A}^\star[t]$ and $\mathcal{I}(u) =_\alpha \mathcal{A}^\star[u]$, by Lemma 5(3), $\mathcal{I}(t) \neq \mathcal{I}(u)$, and thus $\mathcal{I} \models t \not\approx u$.

Since by assumption F contains only equalities and disequalities, we have $\mathcal{I} \models F$, and since $E \subseteq F$, we conclude that $\mathcal{I} \models E$. □

By Theorems 2, 3, and 6, the calculus is sound and complete for the universal theory of (co)datatypes. We can rightly call it a decision procedure for that theory. The proof of solution soundness is constructive in that it provides a method for constructing a model for a saturated configuration, by means of the mapping $\mathcal{A}^\star$.

4 Implementation as a Theory Solver in CVC4

The decision procedure was presented at a high level of abstraction, omitting quite a few details. This section describes the main aspects of the implementation within the SMT solver CVC4: the integration into CDCL(T) [7], the extension to quantified formulas, and some of the optimizations.

The decision procedure is implemented as a theory solver of CVC4, that is, a specialized solver for determining the satisfiability of conjunctions of literals for its theory. Given a theory $T = T_1 \cup \cdots \cup T_n$ and a set of input clauses F in conjunctive normal form, the CDCL(T) procedure incrementally builds partial assignments of truth values to the atoms of F such that no clause in F is falsified. We can regard such a partial assignment as a set M of true literals. By a variant of the Nelson–Oppen method [8,18], each T_i-solver takes as input the union M_i of (1) the purified form of T_i-literals occurring in M, where fresh variables replace terms containing symbols not belonging to T_i; (2) additional (dis)equalities between variables of types not belonging to T_i. Each T_i-solver either reports that a subset C of M_i is T_i-unsatisfiable, in which case $\neg C$ is added to F, adds a clause to F, or does nothing. When M is a complete assignment for F, a theory solver can choose to do nothing only if M_i is indeed T_i-satisfiable.

Assume E is initially the set M_i described above. With each equality $t \approx u$ added to E, we associate a set of equalities from M_i that together entail $t \approx u$, which we call its *explanation*. Similarly, each $\mathcal{A}[x]$ is assigned an explanation—that is, a set of equalities from M_i that entail that the values of $[x]$ in models of E are of the form $\mathcal{A}[x]$. For example, if $x \approx \mathsf{C}(x) \in M_i$, then $x \approx \mathsf{C}(x)$ is an explanation for $\mathcal{A}[x] = \mu\widetilde{x}.\, \mathsf{C}(\widetilde{x})$. The rules of the calculus are implemented as follows. For all rules with conclusion $\bot$, we report the union of the explanations for all premises is $\mathcal{DC}$-unsatisfiable. For Split, we add the exhaustiveness clause $t \approx \mathsf{C}_1\big(\mathsf{s}_1^1(t),\ldots,\mathsf{s}_1^{n_1}(t)\big) \vee \cdots \vee t \approx \mathsf{C}_m\big(\mathsf{s}_m^1(t),\ldots,\mathsf{s}_m^{n_m}(t)\big)$ to F. Decisions on which branch to take are thus performed externally by the SAT solver. All other rules add equalities to the internal state of the theory solver. The rules in phase 1 are performed eagerly—that is, for partial satisfying assignments M—while the rules in phases 2 and 3 are performed only for complete satisfying assignments M.

Before constructing a model for F, the theory solver constructs neither μ-terms nor the mapping $\mathcal{A}$. Instead, $\mathcal{A}$ is computed implicitly by traversing the equivalence classes of E during phase 2. To detect whether Acyclic applies, the procedure considers each equivalence class $[t]$ containing a datatype constructor $\mathsf{C}(t_1,\ldots,t_n)$. It visits $[t_1],\ldots,[t_n]$ and all constructor arguments in these equivalence classes recursively. If while doing so it returns to $[t]$, it deduces that Acyclic applies. To recognize when the precondition of Unique holds, the procedure considers the set S of all codatatype equivalence classes. It simultaneously visits the equivalence classes of arguments of constructor terms in each equivalence class in S, while partitioning S into $S_1,\ldots,S_n$ based on the top-most symbol of constructor terms in these equivalence classes and the equivalence of their arguments

of ordinary types. It then partitions each set recursively. If the resulting partition contains a set S_i containing distinct terms u and v, it deduces that Unique applies to u and v.

While the decision procedure is restricted to universal conjectures, in practice we often want to solve problems that feature universal axioms and existential conjectures. Many SMT solvers, including CVC4, can reason about quantified formulas using incomplete instantiation-based methods [16, 23]. These methods extend naturally to formulas involving datatypes and codatatypes.

However, the presence of quantified formulas poses an additional challenge in the context of (co)datatypes. Quantified formulas may entail an upper bound on the cardinality of an uninterpreted type u. When assuming that u has infinite cardinality, the calculus presented in Section 3 is incomplete since it may fail to recognize cases where Split and Single should be applied. This does not impact the correctness of the procedure in this setting, since the solver is already incomplete in the presence of quantified formulas. Nonetheless, two techniques help increase the precision of the solver. First, we can apply Split to datatype terms whose cardinality depends on the finiteness of uninterpreted types. Second, we can conditionally apply Single to codatatype terms that may have cardinality one. For example, the stream_u codatatype has cardinality one precisely when u has cardinality one. If there exist two equivalence classes $[s]$ and $[t]$ for this type, the implementation adds the clause $(\exists x\, y^u.\ x \not\approx y) \vee s \approx t$ to F.

The implementation of the decision procedure uses several optimizations following the lines of Barrett et al. [2]. Discriminators are part of the signature and not abbreviations. This requires extending the decision procedure with several rules, which apply uniformly to datatypes and codatatypes. This approach often leads to better performance because it introduces terms less eagerly to $\mathcal{T}(E)$. Selectors are collapsed eagerly: If $\mathsf{s}_j^k(t) \in \mathcal{T}(E)$ and $t = \mathsf{C}_j(u_1, \ldots, u_n)$, the solver directly adds $\mathsf{s}_j^k(t) \approx u_k$ to E, whereas the presented calculus would apply Split and Inject before adding this equality. To reduce the number of unique literals considered by the calculus, we compute a normal form for literals as a preprocessing step. In particular, we replace $u \approx t$ by $t \approx u$ if t is smaller than u with respect to some term ordering, replace $\mathsf{C}_j(\bar{t}) \approx \mathsf{C}_{j'}(\bar{u})$ with $\bot$ when $j \neq j'$, replace all selector terms of the form $\mathsf{s}_j^k(\mathsf{C}_j(t_1, \ldots, t_n))$ by t_k, and replace occurrences of discriminators $\mathsf{d}_j(\mathsf{C}_{j'}(\bar{t}))$ by $\top$ or $\bot$ based on whether $j = j'$.

As Barrett et al. [2] observed for their procedure, it is both theoretically and empirically beneficial to delay applications of Split as long as possible. Similarly, Acyclic and Unique are fairly expensive because they require traversing the equivalence classes, which is why they are part of phase 2.

5 Evaluation on Isabelle Problems

To evaluate the decision procedure, we generated benchmark problems from existing Isabelle formalizations using Sledgehammer [3]. We included all the theory files from the Isabelle distribution (Distro, 879 goals) and the *Archive of Formal Proofs* (AFP, 2974 goals) [10] that define codatatypes falling within the supported fragment. We added two unpublished theories about Bird and Stern–Brocot trees by Peter Gammie and Andreas Lochbihler (G&L, 317 goals). To exercise the datatype support, theories

	Distro		AFP		G&L		Overall	
	CVC4	Z3	CVC4	Z3	CVC4	Z3	CVC4	Z3
No (co)datatypes	221	209	775	777	52	51	1048	1037
Datatypes without Acyclic	227	–	780	–	52	–	1059	–
Full datatypes	227	213	786	791	52	51	1065	1055
Codatatypes without Unique	222	–	804	–	56	–	1082	–
Full codatatypes	223	–	804	–	**59**	–	1086	–
Full (co)datatypes	**229**	–	**815**	–	**59**	–	**1103**	–

Table 1. Number of solved goals for the three benchmark suites

about lists and trees were added to the first two benchmark sets. The theories were selected before conducting any experiments. The experimental data are available online.[1]

For each goal in each theory, Sledgehammer selected about 16 lemmas, which were monomorphized and translated to SMT-LIB 2 along with the goal. The resulting problem was given to the development version of CVC4 and to Z3 4.3.2 for comparison, each running for up to 60 s on the StarExec cluster [25]. Problems not involving any (co)datatypes were left out. Due to the lack of machinery for reconstructing inferences about (co)datatypes in Isabelle, the solvers are trusted as oracles.

The development version of CVC4 was run on each problem several times, with the support for datatypes and codatatypes either enabled or disabled. The contributions of the acyclicity and uniqueness rules were also measured. Even when the decision procedure is disabled, the problems may contain basic lemmas about constructors and selectors, allowing some (co)datatype reasoning.

The results are summarized in Table 1. The decision procedure makes a difference across all three benchmark suites. It accounts for an overall success rate increase of over 5%, which is quite significant. The raw evaluation data also suggest that the theoretically stronger decision procedures almost completely subsume the weaker ones in practice: We encountered only one goal (out of 4170) that was solved by a configuration of CVC4 and unsolved by a configuration of CVC4 with more features enabled.

Moreover, every aspect of the procedure, including the more expensive rules, make a contribution. Three proofs were found thanks to the acyclicity rule and four required uniqueness. Among the latter, three are simple proofs of the form **by** *coinduction auto* in Isabelle. The fourth proof, by Gammie and Lochbihler, is more elaborate:

lemma *num_mod_den_unique*: $x = \mathsf{Node}\ 0\ \mathsf{num}\ x \Longrightarrow x = \mathsf{num\_mod\_den}$
proof (*coinduction arbitrary*: *x rule*: *tree.coinduct_strong*)
  case (*Eq_tree x*) **show** *?case*
    by (*subst* (1 2 3 4) *Eq_tree*) (*simp add*: *eqTrueI*[OF *Eq_tree*])
qed

where num_mod_den is defined as $\mathsf{num\_mod\_den} = \mathsf{Node}\ 0\ \mathsf{num}\ \mathsf{num\_mod\_den}$.

[1] `http://lara.epfl.ch/~reynolds/CADE2015-cdt/`

6 Conclusion

We introduced a decision procedure for the universal theory of datatypes and codatatypes. Our main contribution has been the support for codatatypes. Both the metatheory and the implementation in CVC4 rely on μ-terms to represent cyclic values. Although this aspect is primarily motivated by codatatypes, it enables a uniform account of datatypes and codatatypes—in particular, the acyclicity rule for datatypes exploits μ-terms to detect cycles. The empirical results on Isabelle benchmarks confirm that CVC4's new capabilities improve the state of the art.

This work is part of a wider program that aims at enriching automatic provers with high-level features and at reducing the gap between automatic and interactive theorem proving. As future work, it would be useful to implement proof reconstruction for (co)datatype inferences in Isabelle. CVC4's finite model finding capabilities [22] could also be interfaced for generating counterexamples in proof assistants; in this context, the acyclicity and uniqueness rules are crucial to exclude spurious countermodels. Finally, it might be worthwhile to extend SMT solvers with dedicated support for (co)recursion.

Acknowledgment. We owe a great debt to the development team of CVC4, including Clark Barrett and Cesare Tinelli, and in particular Morgan Deters, who jointly with the first author developed the initial version of the theory solver for datatypes in CVC4. Our present and former bosses, Viktor Kuncak, Stephan Merz, Tobias Nipkow, Cesare Tinelli, and Christoph Weidenbach, have either encouraged the research on codatatypes or at least benevolently tolerated it, both of which we are thankful for. Peter Gammie and Andreas Lochbihler provided useful benchmarks. Andrei Popescu helped clarify our thoughts regarding codatatypes and indicated related work. Dmitriy Traytel took part in discussions about degenerate codatatypes. Pascal Fontaine, Andreas Lochbihler, Andrei Popescu, Christophe Ringeissen, Mark Summerfield, Dmitriy Traytel, and the anonymous reviewers suggested many textual improvements. The second author's research was partially supported by the Deutsche Forschungsgemeinschaft project "Den Hammer härten" (grant NI 491/14-1) and the Inria technological development action "Contre-exemples Utilisables par Isabelle et Coq" (CUIC).

References

[1] Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011)

[2] Barrett, C., Shikanian, I., Tinelli, C.: An abstract decision procedure for satisfiability in the theory of inductive data types. J. Satisf. Boolean Model. Comput. 3, 21–46 (2007)

[3] Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. J. Autom. Reasoning 51(1), 109–128 (2013)

[4] Blanchette, J.C., Nipkow, T.: Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 131–146. Springer, Heidelberg (2010)

[5] Carayol, A., Morvan, C.: On rational trees. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 225–239. Springer, Heidelberg (2006)

[6] Djelloul, K., Dao, T., Frühwirth, T.W.: Theory of finite or infinite trees revisited. Theor. Pract. Log. Prog. 8(4), 431–489 (2008)

[7] Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): Fast decision procedures. In: Alur, R., Peled, D. (eds.) CAV 2004. LNCS, vol. 3114, pp. 175–188. Springer, Heidelberg (2004)
[8] Jovanović, D., Barrett, C.: Sharing is caring: Combination of theories. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS, vol. 6989, pp. 195–210. Springer, Heidelberg (2011)
[9] Kersani, A., Peltier, N.: Combining superposition and induction: A practical realization. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) FroCoS 2013. LNCS, vol. 8152, pp. 7–22. Springer, Heidelberg (2013)
[10] Klein, G., Nipkow, T., Paulson, L. (eds.): Archive of Formal Proofs. http://afp.sf.net/
[11] Kozen, D.: Results on the propositional μ-calculus. Theor. Comput. Sci. 27, 333–354 (1983)
[12] Leino, K.R.M., Moskal, M.: Co-induction simply—Automatic co-inductive proofs in a program verifier. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 382–398. Springer, Heidelberg (2014)
[13] Leroy, X.: A formally verified compiler back-end. J. Autom. Reasoning 43(4), 363–446 (2009)
[14] Lochbihler, A.: Verifying a compiler for Java threads. In: Gordon, A.D. (ed.) ESOP 2010. LNCS, vol. 6012, pp. 427–447. Springer, Heidelberg (2010)
[15] Lochbihler, A.: Making the Java memory model safe. ACM Trans. Program. Lang. Syst. 35(4), 12:1–65 (2014)
[16] de Moura, L., Bjørner, N.: Efficient E-matching for SMT solvers. In: Pfenning, F. (ed.) CADE-21. LNCS, vol. 4603, pp. 183–198. Springer, Heidelberg (2007)
[17] de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
[18] Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. 1(2), 245–257 (1979)
[19] Pham, T., Whalen, M.W.: RADA: A tool for reasoning about algebraic data types with abstractions. In: Meyer, B., Baresi, L., Mezini, M. (eds.) ESEC/FSE '13. pp. 611–614. ACM (2013)
[20] Reynolds, A., Blanchette, J.C.: A decision procedure for (co)datatypes in SMT solvers. Tech. report, http://lara.epfl.ch/~reynolds/CADE2015-cdt/ (2015)
[21] Reynolds, A., Kuncak, V.: Induction for SMT solvers. In: D'Souza, D., Lal, A., Larsen, K.G. (eds.) VMCAI 2015. LNCS, vol. 8931, pp. 80–98. Springer, Heidelberg (2014)
[22] Reynolds, A., Tinelli, C., Goel, A., Krstić, S., Deters, M., Barrett, C.: Quantifier instantiation techniques for finite model finding in SMT. In: Bonacina, M.P. (ed.) CADE-24. LNCS, vol. 7898, pp. 377–391. Springer, Heidelberg (2013)
[23] Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in SMT. In: FMCAD 2014. pp. 195–202. IEEE (2014)
[24] Rutten, J.J.M.M.: Universal coalgebra—A theory of systems. Theor. Comput. Sci. 249, 3–80 (2000)
[25] Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 367–373. Springer, Heidelberg (2014)
[26] Suter, P., Köksal, A.S., Kuncak, V.: Satisfiability modulo recursive programs. In: Yahav, E. (ed.) SAS 2011. LNCS, vol. 6887, pp. 298–315. Springer, Heidelberg (2011)
[27] Wand, D.: Polymorphic+typeclass superposition. In: de Moura, L., Konev, B., Schulz, S. (eds.) PAAR 2014 (2014)
[28] Weber, T.: SAT-Based Finite Model Generation for Higher-Order Logic. Ph.D. thesis, Technische Universität München (2008)